Making the Diffie-Hellman Protocol Identity-Based

Dario Fiore[1]* and Rosario Gennaro[2]

[1] Dipartimento di Matematica ed Informatica – Università di Catania, Italy. fiore@dmi.unict.it
[2] IBM T.J. Watson Research Center – Hawthorne, New York 10532. rosario@us.ibm.com

Abstract. This paper presents a new identity based key agreement protocol. In id-based cryptography (introduced by Adi Shamir in [23]) each party uses its own identity as public key and receives his secret key from a master Key Generation Center, whose public parameters are publicly known.
The novelty of our protocol is that it can be implemented over any cyclic group of prime order, where the Diffie-Hellman problem is supposed to be hard. It does not require the computation of expensive bilinear maps, or additional assumptions such as factoring or RSA.
The protocol is extremely efficient, requiring only twice the amount of bandwith and computation of the *unauthenticated* basic Diffie-Hellman protocol. The design of our protcol was inspired by MQV (the most efficient authenticated Diffie-Hellman based protocol in the public-key model) and indeed its performance is competitive with respect to MQV (especially when one includes the transmission and verification of certificates in the MQV protocol, which are not required in an id-based scheme). Our protocol requires a single round of communication in which each party sends only 2 group elements: a very short message, especially when the protocol is implemented over elliptic curves.
We provide a full proof of security in the Canetti-Krawczyk security model for key exchange, including a proof that our protocol satisfies additional security properties such as perfect forward secrecy, and resistance to reflection and key-compromise impersonation attacks.

1 Introduction

Identity-based cryptography was introduced in 1984 by Adi Shamir [23]. The goal was to simplify the management of public keys and in particular the association of a public key to the identity of its holder. Usually such binding of a public key to an identity is achieved by means of *certificates* which are signed statements by trusted third parties that a given public key belongs to a user. This requires users to obtain and verify certificates whenever they want to use a specific public key, and the management of public key certificates remains a technically challenging problem.

Shamir's idea was to allow parties to use their identities as public keys. An id-based scheme works as follows. A trusted *Key Generation Center* (KGC) generates a master public/secret key pair, which is known to all the users. A user with identity ID receives from the KGC a secret key S_{ID} which is a function of the string ID and the KGC's secret key (one can think of S_{ID} as a signature by the KGC on the string ID). Using S_{ID} the user can then perform cryptographic tasks. For example in the case of *id-based encryption* any party can send an encrypted message to the user with identity ID using the string ID as a public key and the user (and only the user and the KGC) will be able to decrypt it using S_{ID}. Note that the sender can do this even if the recipient has not obtained yet his secret key from the KGC. All the sender needs to know is the recipient's identity and the public parameters of the KGC. This is the major advantage of id-based encryption.

Id-Based Key Agreement and its Motivations. This paper is concerned with the task of *id-based key agreement.* Here two parties Alice and Bob, with identities A, B and secret keys S_A, S_B respectively, want to agree on a common shared key, in an *authenticated* manner (i.e. Alice must be

* Work done while visiting NYU and IBM Research.

sure that once the key is established, only Bob knows it – and viceversa). Since key agreement is inherently an interactive protocol (both parties are "live" and ready to establish a session) there is a smaller gain in using an id-based solution: indeed certificates and public keys can be easily sent as part of the protocol communication.

Yet the ability to avoid sending and verifying public key certificates is a significant practical management advantage (see e.g. [26]). Indeed known shortcomings of the public key setting are the requirement of centralized certification authorities, the need for parties to cross-certify each other (via possibly long certificate chains), and the management of some form of large-scale coordination and communication (possibly on-line) to propagate certificate revocation information. Identity-based schemes significantly simplify identity management by bypassing the certification issues. All a party needs to know in order to generate a shared key is its own secret key, the public information of the KGC, and the identity of the communication peer (clearly, the need to know the peer's identity exists in any scheme including a certificate-based one).

Another advantage of identity-based systems is the versatility with which identities may be chosen. Since identities can be arbitrary string, they can be selected according to the the function and attributes of the parties (rather than its actual "name"). For example in vehicular networks a party may be identified by its location ("the checkpoint at the intersection of a and b") or in military applications a party can be identified by its role ("platoon x commander"). This allows parties to communicate securely with the intended recipient even without knowing its "true" identity but simply by the definition of its function in the network.

Finally, identities can also include additional attributes which are temporal in nature: in particular an "expiration date" for an identity makes revocation of the corresponding secret key much easier to achieve.

The reasons described above, explain why id-based KA protocols are very useful in many systems where bandwith and computation are at a premium (e.g. sensor networks), and also in ad-hoc networks where large scale coordination is undesirable, if not outright impossible. Therefore it is an important question to come up with very efficient and secure id-based KA protocols.

Previous work on id-based key agreement. Following Shamir's proposal of the concept of id-based cryptography, some early proposals for id-based key agreement appeared in the literature: we refer in particular to the works of Okamoto [19] (later improved in [20]) and Gunther [13]. A new impetus to this research area came with the breakthrough discovery of bilinear maps and their application to id-based encryption in [2]: starting with the work of Sakai *et al.* [24] a large number of id-based KA protocols were designed that use pairings as tool. We refer the readers to [3] and [8] for surveys of these pairing-based protocols.

The main problem with the current state of the art is that many of these protocols lack a proof of security, and some have even been broken. Indeed only a few (e.g., [5, 27]) have been proven according to a formal definition of security.

Our Contribution. By looking at prior work we see that provably secure id-based KA requires either groups that admit bilinear maps [5, 27], or to work over a composite RSA modulus [20].

This motivated us to ask the question if an efficient and provably secure id-based KA could be found that can be implemented over *any* cyclic group in which the Diffie-Hellman problem is supposed to be hard. The advantages of such a KA protocol would be several, in particular: (i) it would avoid the use of computationally expensive pairing computations; (ii) it could be implemented over much smaller groups (since we could use 'regular' elliptic curves, rather than the supersingular ones that admit pairings, or the group Z_N^* for a composite N needed for Okamoto-Tanaka).

Our new protocol presented in this paper, achieves all these features.

In addition our new protocol is extremely efficient, requiring an amount of bandwith and computation similar to the *unauthenticated* basic Diffie-Hellman protocol. Indeed our protocol requires a single round of communication in which each party sends just two group elements (as opposed to one in the Diffie-Hellman protocol). Each party must compute four exponentiations to compute the session key (as opposed to two in the Diffie-Hellman protocol).

A similar favorable comparison holds with the Okamoto-Tanaka protocol in [20]. While that protocol requires only two exponentiations, it does work over Z_N^* therefore requiring the use of a much larger group size, which almost totally absorbs the computational advantage, and immediately implies a much larger bandwith requirement. The protocol is also competitive vs. KA protocols (such as MQV) in the public key model, particularly when one includes the transmission and verification of certificates which are not required in an id-based scheme. Detailed comparisons to other protocols in the literature are discussed in Section 5.

We present a full proof of security of our protocol in the Canetti-Krawczyk security model. Our results hold in the random oracle model, under the Gap Diffie-Hellman Assumption (see details below). Our protocol can be proven to satisfy additional desirable security properties such as perfect forward secrecy[3], and resistance to reflection and key-compromise impersonation attacks.

Our Approach. The first direction we took in our approach was to attempt to analyze the protocol by Gunther [13], which also works over any cyclic group where the Diffie-Hellman problem is assumed to be hard, but was presented without a formal proof of security. In the [13] protocol the KGC provides each user with an ElGamal signature of its identity. The parties then use these signatures, and the KGC's public key to authenticate a basic Diffie-Hellman exchange. The protocol requires two rounds (four messages) of interaction, and several exponentiations. For these performance reasons, the protocol was already not a very attractive candidate. Moreover we were not able to prove the protocol, as described in [13]. However we devised some modifications to the protocol, and prove such modified version under some strong non-standard assumption, such as *knowledge of exponent* [9], in the random oracle model.

We then turned our attention to a modification of Gunther's protocol presented by Saeednia [22], but also without a proof of security. The protocol in [22] utilizes a variation of ElGamal signatures for the KGC to issue secret keys to the users. This allows a single round protocol, which still however requires several exponentiations. Moreover, here too we were not able to prove the protocol as stated in [22]. Again, we were able to find several modifications to this protocol and find a proof for this modified version under the Gap-DH assumption, in the random oracle model.

For lack of space these modified Gunther and Saeednia protocols and their proofs are presented in Appendix D.

Our protocol improves over these two protocols by using Schorr's signatures [25], rather than ElGamal, to issue secret keys to the users. The simpler structure of Schnorr's signatures permits a much more efficient computation of the session key, resulting in less exponentiations and a single round protocol. Our approach was inspired by the way the MQV protocol achieves *implicit authentication* of the session key. Indeed our protocol can be seen as an id-based version of the MQV protocol.

[3] We can prove PFS only in the case the adversary was passive in the session that he is attacking – though he can be active in other sessions. As proven by Krawczyk in [17], this is the best that can be achieved for 1-round protocols with *implicit* authentication, such as ours.

Probably for this reason, the proof of our protocol heavily uses techniques developed by Krawczyk in the security proof of the HMQV protocol (the provably secure version of MQV) [17]. In particular our proof uses the notion of *challenge-response signatures* introduced in [17], though with substantial changes to adapt it to the identity-based scenario.

2 Preliminaries

In this section we present some standard definitions needed in the rest of the paper.

Let $\mathbb{N}$ the set of natural numbers. We will denote with $\ell \in \mathbb{N}$ the security parameter. The partecipants to our protocols are modeled as probabilistic Turing machines whose running time is bounded by some polynomial in ℓ. If S is a set, we denote with $s \overset{\$}{\leftarrow} S$ the process of selecting an element uniformly at random from S.

Definition 1 (Negligible function). *A function $\epsilon(\ell)$ is said to be negligible if for every polynomial $p(\ell)$ there exists a positive integer $c \in \mathbb{N}$ such that $\forall \ell > c$ we have $\epsilon(\ell) < 1/p(\ell)$.*

Our results are proved secure in the Canetti-Krawczyk (CK) [6, 7] model for key agreement, adpated to the identity-based setting. For lack of space we postpone the description of the model in Appendix B.

2.1 Computational Assumptions

In the following assume $\mathbb{G}$ to be a cyclic multiplicative group of order q where q is a ℓ-bit long prime. We assume that there are efficient algorithms to performe multiplication and membership test in $\mathbb{G}$. Finally we denote with g a generator of $\mathbb{G}$.

Assumption 1 (Computational Diffie-Hellman [10]) *We say that the Computational Diffie-Hellman (CDH) Assumption (for $\mathbb{G}$ and g) holds if for any probabilistic polynomial time adversary $\mathcal{A}$ the probability that $\mathcal{A}$ on input $(\mathbb{G}, g, g^u, g^v)$ outputs W such that $W = g^{uv}$ is negligible in ℓ. The probability of success of $\mathcal{A}$ is taken over the uniform random choice of $u, v \in \mathbb{Z}_q$ and the coin tosses of $\mathcal{A}$.*

The CDH Assumption has a *Decisional* version in which no adversary can actually *recognize* the value g^{uv} when given g^u, g^v. In our proofs we are going to need the ability to perform such decisions, while still assuming that the CDH holds. The assumption below basically says that the CDH Assumption still holds in the presence of an oracle $\mathcal{O}$ that solves the decisional problem.

Assumption 2 (Gap-DH Assumption) *We say that the Gap-DH Assumption holds (for $\mathbb{G}$ and g) if the CDH Assumption holds even in the presence of an oracle $\mathcal{O}$ that on input three elements $U = g^u, V = g^v, W = g^w$ in the group generated by g, output "yes" if and only if $W = g^{uv}$.*

The oracle $\mathcal{O}$ for the Decisional DH problem exists for some groups $\mathbb{G}$, e.g. the ones that admit a bilinear map. We stress, however that we need the oracle *only for the proof of security, and it is not needed in the execution of the protocol by the real-life parties.* This means that we can efficiently implement our protocol over any cyclic group $\mathbb{G}$.

The question, then, is the real-life meaning of a proof under the Gap-DH assumption. We prove the security of our protocol under the Gap-DH assumption, which means that if a successful adversary can be constructed one of two things must be true:

1. either the CDH Assumption is false
2. or we have a proof that the hardness of the Decisional problem is implied by the CDH Assumption (in other words the CDH and DDH Assumptions are equivalent). Indeed in this case the CDH holds, and the protocol is insecure, which means that the oracle $\mathcal{O}$ cannot exists (if it existed, given that the CDH holds, the protocol should be secure).

3 The new protocol IB-KA

This section presents our new identity-based key-agreement protocol IB-KA whose security relies on the Gap-DH assumption in the random oracle model.

Protocol setup. The Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q (where q is ℓ-bits long), a random generator $g \in \mathbb{G}$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_2 : \mathbb{Z}_q \times \mathbb{Z}_q \to \{0,1\}^\ell$. Then it picks a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $y = g^x$. Finally the KGC outputs the public parameters $MPK = (\mathbb{G}, g, y, H_1, H_2)$ and keeps the master secret key $MSK = x$ for itself.

Key Derivation. A user with identity ID receives, as its secret key, a Schnorr's signature [25] of the message $m = ID$ under public key y. More specifically, the KGC after verifying the user's identity, creates the associated secret key as follows. First it picks a random $k \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $r_{ID} = g^k$. Then it uses the master secret key x to compute $s_{ID} = k + H_1(ID, r_{ID})x$. (r_{ID}, s_{ID}) is the secret key returned to the user. The user can verify the correctness of its secret key by using the public key y and checking the equation $g^{s_{ID}} \overset{?}{=} r_{ID} \cdot y^{H_1(ID, r_{ID})}$.

A protocol session. Let's assume that Alice wants to establish a session key with Bob. Alice owns secret key (r_A, s_A) and identity A while Bob has secret key (r_B, s_B) and identity B.

Alice selects a random $t_A \overset{\$}{\leftarrow} \mathbb{Z}_q$, computes $u_A = g^{t_A}$ and sends the message $\langle A, r_A, u_A \rangle$ to Bob. Analogously Bob picks a random $t_B \overset{\$}{\leftarrow} \mathbb{Z}_q$, computes $u_B = g^{t_B}$ and sends $\langle B, r_B, u_B \rangle$ to Alice. After the parties have exchanged these two messages, they are able to compute the same session key $Z = H_2(z_1, z_2)$. In particular Alice computes

$$z_1 = (u_B r_B y^{H_1(B, r_B)})^{t_A + s_A} \quad \text{and} \quad z_2 = u_B^{t_A}.$$

On the other hand Bob computes

$$z_1 = (u_A r_A y^{H_1(A, r_A)})^{t_B + s_B} \quad \text{and} \quad z_2 = u_A^{t_B}.$$

It is easy to see that both the parties are computing the same values $z_1 = g^{(t_A + s_A)(t_B + s_B)}$ and $z_2 = g^{t_A t_B}$.

Theorem 3. *Under the Gap-DH Assumptions, if we model H_1 and H_2 as random oracles, then protocol* IB-KA *is a secure identity-based key agreement protocol.*

4 Proof of security

We prove the theorem by a usual reduction argument. We show how to reduce the existence of an adversary breaking the protocol into an algorithm (i.e. the simulator) that is able to break the CDH Assumption with non-negligible probability. The adversary is modeled as a CK attacker (see Appendix B for details): in particular it will choose a test session among the complete and unexposed sessions and will try to distinguish between its real session key and a random one.

The proof of Theorem 3 considers two different cases: if the test session has a matching session or not. Recall that if two sessions are matching then there are two peers that have identical views of the joint session. So a test session with a matching session implies that the adversary was passive, and not injecting any messages during the test session. The case in which the test session does not have a matching session, instead corresponds to the case in which the adversary is actively injecting messages and trying to impersonate a honest user.

Either case will be reduced to an algorithm that solves the CDH problem. Only in the case the test session does *not* have a matching session we make use of the Gap-DH Assumption, and therefore need an oracle $\mathcal{O}$ to solve the DDH.

In the proof the simulator guesses a-priori the session that will be chosen by the adversary as the test session. More precisely it guesses the owner of the session and the order number of the test session among all the sessions hold by that party. The probability of a correct guess is $1/n$ where n is an upper bound to the number of sessions run by the adversary.

4.1 Passive Adversary during the test session

For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that is able to break the protocol $\mathsf{IB\text{-}KA}$ with non-negligible advantage ϵ in the case the test session has a matching session. Let n be an upper bound to the number of sessions of the protocol run by $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by the adversary to the random oracles H_1, H_2 respectively. We show how to build a simulator S that uses $\mathcal{A}$ to solve the CDH problem with probability at least ϵ/nQ_2. S receives in input a tuple $(\mathbb{G}, g, U, V)$ where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$. The simulator plays the role of the CDH solver and its goal it to compute the value $W = g^{uv}$. S sets up a simulated execution of the protocol, with simulated KGC, users and sessions.

First of all S sets up the public parameters of the protocol simulating the KGC. So it chooses a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$.

Since S knows the master secret key x, it can simulate the KGC in full, and give secret keys to all the parties in the network, including answering private key queries from the adversary.

At the beginning of the game S guesses the test session and its peers Alice and Bob.

Sessions different from the test session are easily simulated since S knows all the informations needed to compute the session keys and answer any query (including session key queries) from the adversary.

We now show how to simulate the test session in order to extract $W = g^{uv}$ from the adversary. Since the test session has a matching session, this means that the parties (i.e. the simulator in this case) choose the messages exchanged in the test session.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity information and the secret keys of Alice and Bob respectively (S knows these values). The simulator sets Alice's message as $(A, r_A, u_A = U)$ while the one from Bob is $(B, r_B, u_B = V)$. S is implicitly setting $t_A = u, t_B = v$. In this case the correct session key is $Z = H_2(g^{(s_A+u)(s_B+v)}, g^{uv})$. Since H_2 is modeled as a random oracle, if the adversary has success into distinguishing Z from a random value, it must have queried H_2 on the correct input $(z_1 = g^{(s_A+u)(s_B+v)}, z_2 = g^{uv})$. Thus S can choose a random value among all the queries that it received from the adversary. Since the number of queries Q_2 is polynomially bounded, the simulator can find the correct $z_2 = W$ with non-negligible probability ϵ/nQ_2. This completes the proof of this case[4].

4.2 Exponential Challenge-Response Signatures

Before proving the case in which the adversary is active during the test session, we introduce the notion of *Challenge-Response Signatures* which are instrumental in obtaining a reduction for that case.

Exponential Challenge-Response (XCR) Signatures were introduced by Krawczyk in [17] as a building block for the proof of the HMQV key-exchange protocol. Roughly speaking XCR signatures consist of an interactive signing process where the recipient of a signature gives a *challenge* to the signer and the latter generates the signature on a message with respect to this challenge. Only who creates the challenge will be able to verify the correctness of the purposed signature.

We propose a variation of the XCR signature scheme given by Krawczyk in [17] which uses a *double* challenge instead of a single one.

2XCR signature scheme Our Exponential Challenge-Response Signature scheme 2XCR is defined by three algorithms: $XKG, XSIG, XVER$.

The key generation algorithm $XKG(1^\ell)$ takes as input the security parameter ℓ, chooses a ℓ-bit prime q and a group $\mathbb{G}$ of order q. Then it picks a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and outputs the verification key $y = g^x$ and the secret key x.

A user wishing to receive a signature, first generates two challenge values $T_1 = g^{t_1}$ and $T_2 = g^{t_2}$ for random $t_1, t_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and gives (T_1, T_2) to the signer. To produce a signature on a message m the signer runs $XSIG(x, m, T_1, T_2)$ which chooses random $k, t \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $w = g^t, r = g^k$ and $s = k + H(m, r)x$ where $H : \{0,1\}^* \to \mathbb{Z}_q$ is an hash function. Finally it computes $z_1 = (T_1T_2)^{s+t}, z_2 = T_2^t$ and outputs the pair (w, r, z_1, z_2) as the signature for m.

The recipient can verify the signature (w, r, z_1, z_2) with respect to a message m, public key y, and challenge $(T_1 = g^{t_1}, T_2 = g^{t_2})$ for which it knows t_1, t_2 by checking if $z_1 \overset{?}{=} (wry^{H(m,r)})^{t_1+t_2}$ and $z_2 \overset{?}{=} w^{t_2}$ (this is the verification algorithm $XVER$).

A novel property of this scheme (not present in Krawczyk's scheme) which is important in our protocol is that a signer can pre-compute (or receive) a *signing-token* $(r = g^k, s = k + H(m, r)x)$ for a message m and then is able to generate signatures on that message for every challenge (T_1, T_2) (i.e. to compute the signature it chooses $t \overset{\$}{\leftarrow} \mathbb{Z}_q$ and outputs $(w, r, z_1 = (T_1T_2)^{s+t}, z_2 = T_2^t)$).

Definition 2 (Security of 2XCR). *The* 2XCR *signature scheme is said to be secure if any PPT forger algorithm $\mathcal{F}$ has at most negligible probability of winning the game below.*

Setup The Challenger runs the key generation algorithm $(y, x) \leftarrow XKG(1^\ell)$, generates a challenge $(T_1 = g^{t_1}, T_2 = g^{t_2})$ for random $t_1, t_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and runs the forger $\mathcal{F}$ on input (y, T_1, T_2).

Signing queries $\mathcal{F}$ is provided access to a *token-signing oracle* $TokSig(x, \cdot)$ that, given in input a message m, outputs a *signing-token* (r, s) for m.

[4] We could give the simulator access to the Gap-DH oracle $\mathcal{O}$, and then S could use it to "test" all queries to H_2 to find the correct W. The reduction would be tighter (removing the factor of Q_2^{-1} from the success probability) but would require the Gap-DH Assumption also in this case.

Forgery The forger wins the game if it outputs a tuple $(m^*, w^*, r^*, z_1^*, z_2^*)$ such that: (i) (w^*, r^*, z_1^*, z_2^*) is a valid signature with respect to the message m^* and the challenge (T_1, T_2) and (ii) m^* was not queried to the oracle $TokSig(x, \cdot)$.

We point out that this definition of security is slightly different from the one given in [17]. Apart having the "double" challenge, we provide the forger with access to the more generic oracle $TokSig(x, \cdot)$ instead of an oracle that outputs signatures when queried on a message-challenge pair. The following theorem proves the security of the scheme (for lack of space its proof appears in Appendix C.1).

Theorem 4. *The* 2XCR *signature scheme is secure according to Definition 2 under the CDH Assumption if H is modeled as a random oracle.*

4.3 Shared Challenge Response Signatures

Here we introduce the notion of Shared Challenge Response (SCR) signatures. Let A and B be two parties that share the same signing key. Very informally, an SCR signature of A and B on messages m_1, m_2 respectively is jointly computed by the two parties and cannot be forged by a third party that does not know the secret key.

In some sense the notion of SCR signatures is similar to that one of Dual Challenge-Response (DCR) signatures introduced by Krawczyk in [17] with the difference that a DCR signature is computed by two parties A and B, each with its own secret key.

As in the case of XCR signatures we define the notion of SCR signatures directly by presenting the construction of the scheme.

SCR signature scheme Let $SKG(1^\ell)$ be the key generation algorithm that chooses a ℓ-bit prime q and a group $\mathbb{G}$ of order q. It picks a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and outputs the verification key $y = g^x$ and the secret key x.

Let A and B be two parties that share the signing key x and want to compute a signature on messages m_1, m_2 where m_1 is chosen by A and m_2 is chosen by B. The signature is computed interactively as follows. First A and B generate the values $u_A = g^{t_A}, r_A = g^{k_A}$ and $u_B = g^{t_B}, r_B = g^{k_B}$ respectively. After having exchanged these values and the messages, the signature is defined by u_A, r_A, u_B, r_B and the pair $(z_1 = g^{(t_A+k_A+H(m_1,r_A)x)(t_B+k_B+H(m_1,r_B)x)}, z_2 = g^{t_A t_B})$. In particular A can compute $(z_1 = (u_B r_B y^{H(m_2,r_B)})^{(t_A+k_A+H(m_1,r_A)x)}, z_2 = u_B^{t_A})$ while B computes $(z_1 = (u_A r_A y^{H(m_1,r_A)})^{(t_B+k_B+H(m_2,r_B)x)}, z_2 = u_A^{t_B})$. As they can compute the same signature string, then each party is also able to verify the signature produced by the other party.

We observe that a SCR signature can be seen as a 2XCR signature by party A on message m_1 under challenge $(u_B r_B y^{H(m_2,r_B)}, u_B)$ and at the same time a 2XCR signature by party B on message m_2 under challenge $(u_A r_A y^{H(m_1,r_A)}, u_A)$. In this sense A and B interact with each other to compute a 2XCR signature, each playing both the roles of challenger and signer.

Moreover the SCR scheme has the interesting property that a party can pre-compute (or receive) a *signing-token* $(r = g^k, s = k + H(m,r)x)$ for a message m and then participate with another party choosing m' to the generation of a SCR signature on messages (m, m').

We define the security of the SCR signature scheme as follows.

Definition 3 (Security of the SCR signature scheme). *The SCR signature scheme is said to be secure if any PPT forger algorithm $\mathcal{F}$ has at most negligible probability of winning the game below.*

Setup The Challenger generates a pair of keys $(y, x) \leftarrow SKG(1^\ell)$ and then runs the forger $\mathcal{F}$ on input y.

Signing queries $\mathcal{F}$ is provided access to a *token-signing oracle* $TokSig(x, \cdot)$ that, given in input a message m, outputs a *signing-token* (r, s) for m. If the same message m is queried more than once, the same (r, s) is returned.

Challenge At some point $\mathcal{F}$ outputs a message m_1 as the one for which it wants to produce a forgery. The Challenger returns to $\mathcal{F}$ the signature token $(r_1 = g^{k_1}, s_1)$ for m_1 and a random value $u_1 = g^{t_1}$.

Forgery The forger wins the game if it outputs a tuple $(m_2, u_2, r_2, z_1, z_2)$ such that: (i) $(u_1, r_1, u_2, r_2, z_1, z_2)$ is a valid SCR signature for the messages m_1, m_2 where m_1 is the message related to the Challenger and (ii) m_2 was not queried to the oracle $TokSig(x, \cdot)$.

In the definition above we observe two facts. First, the forger is allowed to choose even the message m_1 related to the Challenger. Second, we consider valid a forgery for messages m_1, m_2 even if $\mathcal{F}$ has asked m_1 to the token signing oracle.

The following theorem proves the security of the SCR signature scheme (for lack of space its proof appears in Appendix C.2).

Theorem 5. *The SCR signature scheme is secure according to Definition 3 under the CDH Assumption if H is modeled as a random oracle.*

The relationship between SCR signatures and IB-KA . It is not hard to see that in the IB-KA protocol, the two parties compute a SCR signature on their identities (seen as messages) and then set the session key as the hash of part of the signature string (z_1, z_2). The parties do not share the signing key (which in this case is the master secret key of the KGC), but each of them has a signing token for its identity. As we have seen, a signing token for a message m is sufficient for participating to the generation of a signature for m without knowing the secret key. The proof below shows that this intuition is correct, by showing that a successfull impersonation attack yields a forger for the SCR scheme.

4.4 Active adversary during the test session

We are now finally able to prove the security of IB-KA in the case the test session does *not* have a matching session. We do this by showing that we can build a forger for the SCR signature scheme above. The construction of this forger requires an oracle that solves the DDH, and therefore the reduction holds under the Gap-DH Assumption.

For sake of contradiction, let $\mathcal{A}$ be a PPT adversary that is able to break the protocol IB-KA within time T with non-negligible advantage ϵ in the case the test session does not have a matching session. Then we show how to exploit such $\mathcal{A}$ to solve the CDH problem in expected polynomial time via the construction of a forger for SCR signatures. $\mathcal{A}$ is assumed to be a probabilistic polynomial time (PPT) Turing machine with random tape ω that during its run asks a polynomial number of queries to the random oracles H_1 and H_2 and to the key derivation oracle.

The simulator is given as input $\mathbb{G}, g, y, H$ where y is the public key of the SCR signature scheme and H is the hash function used in the SCR signature.

First we describe how S simulates the environment for a run of $\mathcal{A}$. S sets the public parameters of the KGC as $MPK = (\mathbb{G}, g, y, H_1, H_2)$ where $(\mathbb{G}, g, y)$ is its input, $H_1 = H$, and H_2 is a random oracle controlled by S as described below. At the beginning of the game the simulator guesses the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

In order to simulate *party corruption* S must provide the adversary with the secret keys of any party the adversary corrupts (except Bob who cannot be corrupted). S does this by querying the "signature token oracle" for the SCR signature scheme. Given in input an identity ID of a user who is not Bob ($ID \neq B$), S queries ID to the signature token oracle and receives r, s such that $g^s = ry^{H_1(ID,r)}$.

For the case of Bob, the simulator simply chooses the r_B component of Bob's private key by picking a random $k_B \overset{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$. We observe that in this case S is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

For each pair (z_1, z_2) received in input, the random oracle H_2 is simulated by choosing a random string $Z \in \{0,1\}^\ell$ and storing each triple (z_1, z_2, Z) in a table $\overline{H_2}$.

First we describe how to simulate sessions different from the test session. Here the main point is that the adversary is allowed to ask session-key queries and thus the simulator must be able to produce the correct session key for each of these sessions.

The simulator has full information about all the users' secret keys except Bob. Therefore S can easily simulate all the protocol sessions that do not include Bob, and answer any of the attacker's queries about these sessions. Hence we concentrate on describing how S simulates interactions with Bob.

Assume that Bob has a session with Charlie (whose identity is the string C). If Charlie is an uncorrupted party this means that S will generate the messages on behalf of him. In this case S knows Charlie's secret key and also has chosen his ephemeral exponent t_C. Thus it is trivial to see that S has enough information to compute the correct session key.

The case when the adversary presents a message $\langle C, r_C, u_C \rangle$ to Bob as coming from Charlie is more complicated. Here is where S makes use of the Gap-DH oracle to answer a session-key query about this session. The simulator replies with a message $\langle B, r_B, u_B = g^{t_B} \rangle$ where t_B is chosen by S. Recall that the session key is $H_2(z_1, z_2)$ where $z_2 = u_C^{t_B}$ which the simulator can compute since it knows t_B. On the other hand $z_1 = g^{(s_C + t_C)(s_B + t_B)}$. Notice that

$$g^{s_C} = r_C y^{H_1(C, r_C)} \quad \text{and } g^{s_B} = r_B y^{H_1(B, r_B)}$$

which can be computed by the adversary. So z_1 is the Diffie-Hellman result of the values $u_C g^{s_C}$ and $u_B g^{s_B}$, therefore the adversary checks if the table $\overline{H_2}$ contains a tuple (z_1, z_2, Z) such that $z_2 = u_C^{t_B}$ and $DH(u_C r_C y^{H_1(C,r_C)}, u_B r_B y^{H_1(B,r_B)}, z_1) =$ "yes''. If S finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \overset{\$}{\leftarrow} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query (z_1, z_2) to H_2, if (z_1, z_2) satisfies the equation above it answers with ζ. This makes oracle's answers consistent.

Now we describe the simulation of the test session between Alice and Bob, If there is no matching session at Bob, it means that the incoming message $\langle B, \rho_B, u_B = g^{t_B} \rangle$ from Bob to Alice is not sent by Bob, but it is originated by the adversary who is trying to impersonate Bob. Notice that the adversary may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which S simulated and for which it knows k_B). The simulator announces that he will forge on message $m_1 = A$, Alice's identity. It then receives A, r_A (which could have been queried before) and $u_A = g^{t_A}$ from the challenger. The simulator sends the message $\langle A, r_A, u_A \rangle$ as coming from Alice. The correct session key for the test session is the hash $Z = H_2(z_1, z_2)$ where $z_1 = (u_A r_A y^{H_1(A,r_A)})^{(t_B + \lambda_B + xH_1(B,\rho_B))}$ and $z_2 = u_A^{t_B}$. If the adversary has success into distinguishing Z from a random value it must necessarily query the correct pair (z_1, z_2) to the random oracle H_2. This means that S can efficiently find the pair (z_1, z_2) in the table $\overline{H_2}$ using the Gap-DH oracle. The tuple $(B, u_B, \rho_B, z_1, z_2)$ is a forgery for the SCR signature scheme.

4.5 Other security properties of IB-KA

In this section we show that the protocol IB-KA satisfies the other security properties described in Appendix B.3.

Resistance to reflection attacks Here we extend the proof of security given in Section 4 to support reflection attacks. We observe that in the case when the test session has a matching session the proof remains valid even if the test session is between Bob and himself.

On the other hand, when there is no matching session we have to show a little modification of the proof. In fact the proof actually does not work when the adversary sends a message with the same value r_B provided by the KGC (for which the simulator knows the discrete logarithm k_B, but cannot compute the corresponding s_B). The issue is that the knowledge of s_B is needed to extract the solution of the CDH problem.

We point out that a reflection attack using a value $\rho_B \neq r_B$ is captured by the current proof. Moreover it is reasonable to assume that a honest party refuses connections from itself that use a "wrong" key.

In this section we show how to adapt the proof in this specific case. In particular, we show that a successful run of the adversary enables the simulator to compute g^{u^2} instead of g^{uv}. As showed in [18] by Maurer and Wolf, such an algorithm can be easily turned into a solver for CDH.

Let us consider the following modification of the proof given in Section 4. If in the test session the adversary sends a message from Bob to Bob of type $\langle ID_B, r_B, u_B = g^{t_B} \rangle$ then the simulator picks a random $e \xleftarrow{\$} \mathbb{Z}_q$ and replies with message $\langle ID_B, r_B, u'_B = U^e \rangle$. We observe that in this case the correct session key is the hash $Z = H_2(z_1, z_2)$ where $z_1 = g^{(k_B + ud^* + ue)(k_B + ud^* + t_B)}$ and $z_2 = g^{uet_B}$. If the adversary has success into distinguishing Z from a random value it must necessarily query the correct pair (z_1, z_2) to the random oracle H_2. This means that S can efficiently find the pair (z_1, z_2) in the table $\overline{H_2}$ using the Gap-DH oracle. Once it has recovered these values, it can compute:

$$g^{u^2} = \left(\frac{z_1}{g^{k_B^2} U^{2k_B d^*} U^{ek_B} u_B^{k_B} z_2 z_2^{d^*/e}} \right)^{\frac{1}{d^*(d^*+e)}}$$

as desired.

Forward secrecy The protocol IB-KA satisfies the notion of *weak forward secrecy* described in Appendix B.3, which means that IB-KA has forward secrecy only against passive attackers. In order to prove this property, we show that it is just captured by the proof given in Section 4.1 in the case when the test session has a matching session. Let Alice and Bob be the two parties involved in the test session. Observe that the simulator knows the master secret key. Hence it can provide the private keys of Alice and Bob to the attacker whenever it asks for party corruption and it is also able to give the master secret key of the KGC. This does not affect the rest of the proof, because if the attacker succeeds into distinguishing the correct session key from a random one, we are still able to use it to solve the CDH problem.

Though weak forward secrecy is a weaker notion of forward secrecy, we stress that it is the best one can achieve in 2-message protocols that are implicitely authenticated (i.e. protocols where the exchanged DH values do not carry authentication information with them, see [17] for details).

Resistance to Key Compromise Impersonation To see that the protocol IB-KA is resistant to KCI attacks it suffices to observe that in the proof given in Section 4.4 (when the test session has no matching session), when the adversary tries to impersonate Bob to Alice, we are able to output Alice's private key whenever it is asked by the adversary. It means that the proof continues to be valid even in this case.

5 Comparisons with other IB-KA protocols

In this section we compare IB-KA with other id-based KA protocols from the literature. In particular, we consider the protocol by Boyd *et al.* [5] (BMP), which is the most efficient (among pairing-based protocols) according to the survey of Chen *et al.* [8] and two protocols proposed very recently by Boyd *et al.* [4] (BCNP1, BCNP2).

BMP is a three-round three-pass protocol that makes use of pairings. It is proved secure in the CK model using random oracles under the Bilinear Diffie-Hellman Assumption, provides weak forward secrecy but it is not resistant to KCI attacks. BCNP1 and BCNP2 are generic constructions based on any CCA-secure IB-KEM. When implemented (as suggested by the authors of [4]) using one of the IB-KEMs by Kiltz [15], Kiltz-Galindo [16] or Gentry [12] they lead to a two-pass single-round protocol with (CK) security in the standard model. BCNP2 provides weak FS and resistance to KCI attacks, while BCNP satisfies only the former property.

For our efficiency comparisons we consider a security parameter of 128 and implementations of BMP, BCNP1 and BCNP2 with Type 3 pairings which are the most efficient pairings for this kind of security level (higher than 80). Our protocol is assumed to be implemented in an elliptic curves group $\mathbb{G}$ with the same security parameter. In this scenario elements of $\mathbb{G}$ and $\mathbb{G}_1$ need 256 bit to be represented, while 512 bits are needed for $\mathbb{G}_2$ elements and 3072 bits for an element of $\mathbb{G}_T$.

We estimate the computational cost of all the protocols using the costs per operation for Type 3 pairings given by Chen *et al.* in [8]. The bandwidth cost is expressed as the amount of data in bits sent by each party to complete a session of the protocol[5].

The results are summarized in Table 1 assuming protocols BCNP1 and BCNP2 to be implemented with Kiltz's IB-KEM (the most efficient for this application according to the work of Boyd *et al.* [4]). We defer to the original papers of BMP [5] and BCNP1, BCNP2 [4] for more details about these costs.

Our protocol has the minimal bandwidth requirement, and its computational efficiency would only be matched by the Okamoto-Tanaka protocol [20]. However we do not consider [20] in our comparisons because we currently do not know a formal proof of security of it.

[5] We do not consider the identity string sent with the messages as it can be implicit and, in any way, appears in all the protocols.

	weak FS	KCI	Standard model	Efficiency	
				Bandwidth	Cost per party
BCNP1	✗	✓	✓	768	56
BCNP2	✓	✓	✓	1024	59
BMP[*1]	✓	✗	✗	512	23
IB-KA	✓	✓	✗	512	6

[*1] While the other protocols are 2-pass and single-round, BMP has 3 rounds.

Table 1. Comparisons between IB-KA protocols.

References

1. M. Bellare and A. Palacio. The Knowledge-of-Exponent Assumptions and 3-round Zero-Knowledge Protocols. *Advances in Cryptology – CRYPTO 2004*, LNCS vol. 3152.
2. Dan Boneh, Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM J. Comput.* 32(3): 586-615 (2003) (Also in CRYPTO 2001.)
3. Colin Boyd, Kim-Kwang Raymond Choo. Security of Two-Party Identity-Based Key Agreement. Mycrypt 2005: 229-243
4. Colin Boyd, Yvonne Cliff, Juan Gonzales Nieto, Kenneth G. Paterson. Efficient One-Round Key Exchange in the Standard Model. *In proceedings of ACISP 2008*, LNCS vol. 5107, pp. 69-83
5. Colin Boyd, Wenbo Mao, Kenneth G. Paterson. Key Agreement Using Statically Keyed Authenticators. *In proceedings of ACNS 2004*, LNCS vol. 3089, pp. 248-262
6. R. Canetti, H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. *Advances in cryptology – EUROCRYPT 2001*, LNCS vol. 2045, pp. 453-474
7. R. Canetti, H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Advances in cryptology – EUROCRYPT 2002*, LNCS vol. 2332, pp. 337-351
8. L. Chen, Z. Cheng, Nigel P. Smart. Identity-based key agreement protocols from pairings. Int. J. Inf. Sec. 6(4): 213-241 (2007)
9. I. Damgård. Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. *Advances in Cryptology – CRYPTO'91*, LNCS vol. 576.
10. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 1976, vol. 22, n. 6 , pp. 644-654
11. A. Fiat and A. Shamir How to Prove Yourself: Practical Solutions of Identification and Signature Problems. *Advances in cryptology – CRYPTO 1986*, LNCS vol. 263, pp. 186-194
12. C. Gentry. Practical Identity-Based Encryption Without Random Oracles. *Advances in cryptology – proceedings of EUROCRYPT 2006*, 2006, LNCS vol. 4004, pp. 494-510
13. Gunther, C.G. An Identity-Based Key-Exchange Protocol. *Advances in cryptology – proceedings of EUROCRYPT 1989*, 1989, LNCS, vol. 434, pp. 29-37.
14. S. Hada and T. Tanaka. On the Existence of 3-round Zero-Knowledge Protocols. *Advances in Cryptology – CRYPTO 1998*, LNCS vol. 1462
15. E. Kiltz. Direct Chosen-Ciphertext Secure Identity-Based Encryption in the Standard Model with short Ciphertexts. Cryptology Eprint Archive, Report 2006/122. http://eprint.iacr.org/2006/122.
16. E. Kiltz and D. Galindo. Direct Chosen-Ciphertext Secure Identity-Based Key Encapsulation Without Random Oracles. Cryptology Eprint Archive, Report 2006/034. http://eprint.iacr.org/2006/034.
17. Hugo Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. *Advances in cryptology – CRYPTO 2005*, LNCS vol. 3621, pp. 546-566
18. U. Maurer and S. Wolf. Diffie-Hellman oracles. *Advances in cryptology – CRYPTO 1996*, LNCS vol. 1109, pp. 268-282
19. E. Okamoto. Key Distribution Systems Based on Identification Information. In Advances in Cryptology, Crypto 1987, pp. 194-202. LNCS Vol. 293/1988.
20. E. Okamoto and K. Tanaka. Key Distribution System Based on Identification. Information. IEEE Journal on Selected Areas in Communications, 7(4):481-485, May 1989.
21. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361-396 (2000).

22. S. Saeednia. Improvement of Gunther's identity-based key exchange protocol. *Electonics Letters* vol. 36, Issue 18, 31 Aug 2000, pp. 1535 - 1536
23. Adi Shamir Identity-Based Cryptosystems and Signature Schemes *Advances in Cryptology – Proceedings of CRYPTO '84*, 1985, pp. 47-53
24. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. *In Symposium on Cryptography and Information Security*, Okinawa, Japan, 2000.
25. C.P. Schnorr. Efficient identification and signatures for smart cards. *Advances in Cryptology – CRYPTO '89*, 1989, LNCS vol. 435, pp. 239-252
26. Smetters, D.K., Durfee, G.: Domain-based Administration of Identity-Based Cryptosystems for Secure E-Mail and IPSEC. In: SSYM 2003: Proceedings of the 12th Conference on USENIX Security Symposium, Berkeley, CA, USA, p. 15. USENIX Association (2003)
27. Y. Wang. Efficient Identity-Based and Authenticated Key Agreement Protocol. Cryptology ePrint Archive, Report 2005/108, 2005. http://eprint.iacr.org/2005/108/.

A The Forking Lemma

The Forking Lemma was proven by Pointcheval and Stern in [21] as a tool to prove the security of signature schemes based on the Fiat-Shamir paradigm. For a message m, such a signature is a tuple (σ_1, h, σ_2). The Forking Lemma states that if this type of signatures satisfies certain simulatability requirements, then if there exist a forger that output a valid signature (σ_1, h, σ_2) for a message m, then there exists another machine that outputs another valid signature for m which has the same σ_1 component.

Theorem 6 (Forking Lemma [21]). *Let $\mathcal{A}$ be a probabilistic polynomial time Turing machine whose input only consists of public data. We denote respectively by Q and R the number of queries that $\mathcal{A}$ can ask to the random oracle and the signing oracle respectively. Assume that, within time T, $\mathcal{A}$ produces, with probability $\epsilon \geq 10(R+1)(R+Q)/2^k$, a valid signature $(m, \sigma_1, h, \sigma_2)$. If the triples (σ_1, h, σ_2) can be simulated without the knowledge of the signing key, with an indistinguishable distribution probability, then there is another machine which has control over the machine obtained from $\mathcal{A}$ replacing interaction with the signer by simulation and produces two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$ in expected polynomial time $T' \leq 120686QT/\epsilon$.*

Roughly speaking the Forking Lemma shows that if an adversary succeeds into finding a forgery against a signature scheme, then a "rewind" of this adversary is likely to succeed into finding a second forgery for the same message with non-negligible probability, with the same first component. More precisely this result applies to signatures of the form $(m, \sigma_1, h, \sigma_2)$ derived via the Fiat-Shamir methodology when the signing oracle can be simulated without knowledge of the secret key and produces signing keys with an indistinguishable distribution probability. In other words Pointcheval and Stern [21] generalized in a non-interactive fashion the technique of rewinding a three-pass honest-verifier zero-knowledge identification protocol. In fact, as described by Fiat and Shamir [11], these protocols can be turned into generic digital signature schemes in the random oracle model.

We apply the Forking Lemma to our Shared Challenge-Response Signature SCR, which although different from a "standard" signature scheme, still satisfies all the requirements of the Forking Lemma.

B Definitions for identity-based key agreement

The security of our protocols is analyzed in a version of the Canetti-Krawczyk (CK) [6, 7] model for key agreement, adapted to the identity-based setting. We present an informal summary of the model and we refer the reader to [6, 7] for details.

An identity-based key-agreement protocol is runned by parties interacting in a network where each party is identified by a unique identity which is publicly known to all the other parties (e.g. Alice's identity is a string ID_A). In addition there exists a trusted entity called *Key Generation Center* (KGC) that generates the public parameters of the system and also issues secret keys to users associated with their public identities, e.g. the KGC generates a secret key SK_A associated to ID_A.

An instance of the protocol is called a *session*. The two parties participating in the session are called its *peers*. Each peer maintains a *session state* which contains incoming and outgoing messages and its random coins. If the session is *completed* then each party outputs a *session key* and erases its session state. A session may also be *aborted*. In this case no session key is generated.

Each party assigns an unique identifier to a session he is participating in. For simplicity, we assume it to be the quadruple $(Alice, Bob, m_{Out}, m_{In})$ where Alice is the identity of the party, Bob its peer, m_{Out} and m_{In} are the outgoing and incoming messages, respectively, for Alice. If Alice holds a session $(Alice, Bob, m_{Out}, m_{In})$ and Bob holds a session $(Bob, Alice, m_{In}, m_{Out})$ then the two sessions are *matching*.

B.1 The adversary

The CK definition models a very realistic adversary which basically controls all communication in the network. In particular it can intercept and modify messages exchanged by parties, delay or block their delivery, inject its own messages, schedule sessions etc. The adversary is allowed to choose the identities of the parties, and obtain private keys from the KGC for identities of its choice.

Finally we allow the adversary to access some of the parties' secret information, via the following attacks: *party corruption*, *state-reveal queries* and *session-key queries*. When an adversary *corrupts* a party, it learns its private information (the private key and all session states and session keys currently stored), and it later controls its actions. In a *state-reveal query* to a party running a session, the adversary learns the session state for that session (since we assume that session states are erased at the end of the session, such query makes sense only against sessions that are still incomplete). Finally a *session-key query* allows the adversary to learn the session key of a complete session. A session is called *exposed* if it or its matching session (if existing) is compromised by one of the attacks above.

B.2 Security Definition

Let $\mathcal{A}$ be a probabilistic polynomial time adversary modeled as described above. Then consider the following experiment running $\mathcal{A}$.

At the beginning of the game the adversary receives in input the public parameters of the system (generated by the KGC) and then can perform all the actions described in the section before.

At some point, $\mathcal{A}$ chooses a *test session* among all the completed and unexposed sessions. We toss a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$ we give $\mathcal{A}$ the session key K_0 of the test session. Otherwise we take a random session key K_1 and provide $\mathcal{A}$ with K_1.

After having received K_b, the adversary can continue to perform its actions against the protocol with the exception that it cannot expose the test session. At the end of the game $\mathcal{A}$ outputs a bit b' as its guess for b.

Definition 4. *An identity-based key-agreement protocol is said to be secure if for any PPT adversary $\mathcal{A}$ the following holds:*

1. *if two uncorrupted parties complete matching sessions then they output the same session key with overwhelming probability;*
2. *the probability that $\mathcal{A}$ guesses the correct* b *in the above experiment is at most* $1/2$ *plus a negligible fraction of the security parameter.*

We define the advantage of $\mathcal{A}$ as $\mathbf{Adv}_{\mathcal{A}}^{IB-KA} = |\Pr[b = b'] - 1/2|$.

B.3 Additional security properties

In addition to the notion of session key security presented above, an identity-based key-agreement protocol should satisfy other important properties: resistance to *reflection attacks*, *forward secrecy* and resistance to *key-compromise impersonation* attacks.

A *reflection attack* occurs when an adversary can compromise a session in which the two parties have the same identity (and the same private key). Though, at first glance, this seems to be only of theoretical interest, there are real-life situations in which this scenario occurs. For example consider the case when Alice is at her office and wants to establish a secure connection with her PC at home, therefore running a session between two computers with the same identity and private key.

We would also like to achieve resistance to *key compromise impersonation* (KCI) attacks. Suppose that the adversary learns Alice's private key. Then, it is trivial to see that this knowledge enables the adversary to impersonate Alice to other parties. A KCI attack can be carried out when the knowledge of Alice's private key allows the adversary to impersonate another party to Alice.

Finally, *Forward secrecy* is probably the most important additional security property we would like to achieve. We say that a KA protocol has forward secrecy, if after a session is completed and its session key erased, the adversary cannot learn it *even if* it corrupts the parties involved in that session. In other words, learning the private keys of parties should not jeopardize the security of past completed sessions.

A relaxed notion of forward secrecy (which we call *weak*) assumes that only past sessions in which the adversary was passive (i.e. did not choose the messages) are not jeopardized.

C Proofs

C.1 Proof of Theorem 4

Let $\mathcal{F}$ be a PPT algorithm that can break the security of 2XCR with non-negligible probability. Then we show how to use the forger $\mathcal{F}$ to build an efficient solver S for the CDH problem. S receives as input $(\mathbb{G}, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$.

The simulator chooses a random $t_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $y = U, T_1 = g^{t_1}, T_2 = V$ and runs the forger $\mathcal{F}$ on input (y, T_1, T_2).

The random oracle H and the token-signing oracle $TokSig$ are simulated as follows. When queried on a message m, S chooses random $s, d \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $r = g^s/y^d$, outputs (r, s) as the signing-token for m and $H(m, r) = d$ as the random oracle response. Note that these tokens are distributed exactly as in the real scheme.

At some point $\mathcal{F}$ will output a forgery $(m^*, w^*, r^*, z_1^*, z_2^*)$ such that m^* was not queried to the token-signing oracle, $z_1^* = (w^* r^* y^{H(m^*,r^*)})^{t_1+v}$ and $z_2^* = (w^*)^v$. Let d^* be the value chosen by the simulator as the output of the query $H(m^*, r^*)$. Given this forgery, S can compute $\tau = \frac{z_1^*}{z_2^*(w^* r^* y^{d^*})^{t_1}} = (r^*)^v W^{d^*}$.

We observe that our scheme and this simulation fit the requirements needed for applying the Forking Lemma [21]. First, our signatures are almost standard Fiat-Shamir type signatures. Second, in our security proof S is able to generate signatures (i.e. signing token) without the knowledge of the secret key and each with an indistinguishable distribution probability. Thus, if we apply the result of the Forking Lemma to our case we obtain that if $\mathcal{F}$ has output a successfull forgery in its first run, then a second run of $\mathcal{F}$ will do so with non-negligible probability (for completeness we recall the statement of the Forking Lemma in Appendix A).

Therefore the simulator runs again $\mathcal{F}$ on the same input (y, T_1, T_2) and with the same random coins as in the previous successful run. The only difference is that $H(m^*, r^*)$ now is answered with a new random value d'^*. Subsequent queries to H are answered with different random values as well. At the end of the second run $\mathcal{F}$ will output a successful forgery $(m^*, w'^*, r^*, z_1'^*, z_2'^*)$ with non-negligible probability. So S can compute $\tau' = \frac{z_1'^*}{z_2'^*(w'^* r^* y^{d'^*})^{t_1}} = r^{*v} W^{d'^*}$ and then extract $W = (\tau/\tau')^{(d^*-d'^*)^{-1}}$.

In conclusion, we showed an efficient algorithm S that is able to solve the CDH problem using an efficient forger $\mathcal{F}$ for the 2XCR signature scheme, which turns out to be secure according to Definition 2.

C.2 Proof of Theorem 5

The proof of this theorem is similar to that of Theorem 4 for the 2XCR scheme.

Let $\mathcal{F}$ be a PPT algorithm that can break the security of SCR with non-negligible probability. Then we show how to use the forger $\mathcal{F}$ to build an efficient solver S for the CDH problem. S receives as input $(\mathbb{G}, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$.

The simulator runs the forger $\mathcal{F}$ on input $y = U$. The random oracle H and the token-signing oracle $TokSig$ are simulated as follows. When queried on a message m, S chooses random $s, d \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $r = g^s/y^d$, outputs (r, s) as the signing-token for m and $H(m, r) = d$ as the random oracle response. Note that these tokens are distributed exactly as in the real scheme.

At some point $\mathcal{F}$ will output a message m_1 and the simulator will respond with $u_1 = V$ and (r_1, s_1) generated as in signing oracle queries. Finally $\mathcal{F}$ will output a forgery $(m_2, u_2, r_2, z_1, z_2)$ such that m_2 was not queried to the token-signing oracle, $z_1 = g^{(v+s_1)(s_2+t_2)}$ and $z_2 = (u_2)^v$. Let d be the value chosen by the simulator as the output of the query $H(m_2, r_2)$. Given this forgery, S can compute $\tau = \frac{z_1}{z_2(u_2 r_2 y^d)^{s_1}} = r_2^v W^d$.

We observe that our scheme and this simulation fit the requirements needed for applying the Forking Lemma [21]. First, our signatures are almost standard Fiat-Shamir type signatures. Second, in our security proof S is able to generate signatures (i.e. signing token) without the knowledge of the secret key and each with an indistinguishable distribution probability. Thus, if we apply the result of the Forking Lemma to our case we obtain that if $\mathcal{F}$ has output a successfull forgery in its first run, then a second run of $\mathcal{F}$ will do so with non-negligible probability (for completeness we recall the statement of the Forking Lemma in Appendix A).

Therefore the simulator runs again $\mathcal{F}$ on the same input y and with the same random coins as in the previous successful run. The only difference is that $H(m_2, r_2)$ now is answered with a new random value d'. Subsequent queries to H are answered with different random values as well. At the end of the second run $\mathcal{F}$ will output a successful forgery $(m_2, u_2', r_2, z_1', z_2')$ with non-negligible probability. Note that in its second run $\mathcal{F}$ may choose a different message m_1', but this does not affect the proof because the simulator always knows the corresponding signing token r_1', s_1'. So S can compute $\tau' = \frac{z_1'}{z_2'(u_2' r_2 y^{d'})^{s_1'}} = r_2^v W^{d'}$ and then extract $W = (\tau/\tau')^{(d-d')^{-1}}$ as desired.

In conclusion, we showed an efficient algorithm S that is able to solve the CDH problem using an efficient forger $\mathcal{F}$ for the SCR signature scheme, which turns out to be secure according to Definition 3.

D Security analysis of related protocols

As an additional contribution of the paper, in this section we present a formal security analysis of two id-based KA protocols that use techniques that inspired our work: the first by Gunther [13] and the second by Saeednia [22] (which is is an improvement of the previous one). In particular we show variants of these protocols that allow to prove their security in the CK model while only an intuition of security was stated in the original works [13, 22].

D.1 Gunther's protocol

We present a slightly different variant of Gunther's protocol [13] which we prove secure under the Gap-DH and KEA assumptions.

The Knowledge of Exponent Assumption (KEA) was first stated by Damgård in [9] and later discussed in [1, 14]. Let $\mathbb{G}$ be a group of prime order q with generator g. Then we say that KEA holds over $\mathbb{G}$ if: for any efficient algorithm $\mathcal{A}$ that on input (g, g^a) outputs a pair (B, C) such that $C = B^a$ there exists an efficient "extractor" algorithm $\mathcal{A}'$ that given the same input of $\mathcal{A}$ outputs (B, C, b) such that $C = B^a$ and $B = g^b$.

The modified protocol is summarized in Figure 1. We recall that the session key in the original protocol was just $z_1 z_2 z_3$ and the key generation process computed the hash only on the identity string $H(ID)$. So what we change is: to hash the session key and include the value r_{ID} when hashing the identity. Since the key derivation process is essentially an El Gamal signature on the identity string, the latter modification follows what Pointcheval and Stern proposed in [21] to prove the security of the El Gamal signature scheme.

The following theorem proves the security of the protocol.

Theorem 7. *If H_1 and H_2 are modeled as random oracles and the Gap-DH and KEA assumptions hold, then Gunther's protocol is secure according to definition Definition 4.*

Proof. As in the proof of IB-KA we distinguish between the case when the adversary chooses a test session that has a matching session and the other case when the test session hash no matching session. In the former we show that we can break directly the CDH Assumption, while the latter requires the Gap-DH and KEA assumptions.

Gunther's protocol

Setting: A Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q together with a random generator $g \in \mathbb{G}$ and an exponent $x \xleftarrow{\$} \mathbb{Z}_q$. KGC publishes $\mathbb{G}, q, g, y = g^x$ and two hash functions H_1, H_2. It distributes to each user with identity U a private key (r_U, s_U) computed as follows: $r_U = g^k, s_U = k^{-1}(H_1(U, r_U) - xr_U) \bmod q$ for random $k \xleftarrow{\$} \mathbb{Z}_q$.

Key agreement: A and B choose ephemeral private exponents t_A, w_A and t_B, w_B, respectively.

$$\begin{tikzcd} A \arrow[rr, "ID_A, r_A"] & & B \\ & & \arrow[ll, "ID_B, r_B"'] \\ \arrow[rr, "u_A = r_B^{t_A}, v_A = g^{w_A}"] & & \\ & & \arrow[ll, "u_B = r_A^{t_B}, v_B = g^{w_B}"'] \end{tikzcd}$$

$$z_1 = u_B^{s_A} \qquad\qquad z_1 = (g^{H_1(ID_A, r_A)}/y^{-r_A})^{t_B}$$

$$z_2 = (g^{H_1(ID_B, r_B)}/y^{-r_B})^{t_A} \qquad\qquad z_2 = u_A^{s_B}$$

$$z_3 = v_B^{w_A} \qquad\qquad z_3 = v_A^{w_B}$$

$$Z = H_2(z_1 z_2 z_3)$$

Fig. 1. A and B share session key Z.

The test session has a matching session For sake of contradiction assume there exists an adversary $\mathcal{A}$ that is able to break with non-negligible advantage the security of Gunther's protocol choosing a test session that has a matching session. Then we can build an efficient algorithm S that can solve the CDH problem with non-negligible probability.

S receives in input a tuple $(\mathbb{G}, q, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$. First S simulates the KGC setting up the public parameters of the protocol. It chooses a random $x \xleftarrow{\$} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, q, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$. Similar work is done for H_2.

As it is described in Appendix B the adversary is allowed to ask the KGC for the secret keys of users of its choice (except for the two parties involved in the test session). Hence S must be able to simulate the key derivation process. If the adversary asks for the secret key of a user, the simulator is always able to respond, since it has chosen the master secret key x by itself.

At the beginning of the game S guesses the test session and its holder (let us call him Bob). Also let Alice be the other party of the session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$. Sessions different from the test session are easily simulated since S knows all the informations needed to compute the session keys and answer to session key queries.

Without loss of generality we assume that the test session is at Bob (thus the corresponding matching session is at Alice). As pointed out in the proof of IB-KA if the sessions at Alice and Bob are matching, it means that the adversary was passive and not injecting any message during the test session. Thus the parties (i.e. the simulator in this case) choose the messages exchanges in this session.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity informations and the secret keys of Alice and Bob respectively. The simulator uses these values to create the first two messages between the parties. To generate the other ones S chooses random $t_A, t_B \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $\langle u_A = r_B^{t_A}, v_A = U\rangle$ and $\langle u_B = r_A^{t_B}, v_B = V\rangle$. Thus S is implicitly setting $w_A = u, w_B = v$. Since H_2 is modeled as a random oracle, if the adversary has success into distinguishing the real session key from a random value, it must have queried H_2 on the correct input $\bar{z} = u_B^{s_A} u_A^{s_B} g^{uv}$. Thus S can choose a random value among all the queries that it received from the adversary and then extract $W = \bar{z}/(u_B^{s_A} u_A^{s_B})$ from it. In conclusion S can find W with non-negligible probability ϵ/nQ_2. This completes the proof of this case.

Remark 1. If we would assume the simulator having access to a Gap-DH oracle $\mathcal{O}$, S might use the oracle to test, for all queries z made by the adversary, if $DH(U, V, z_3) =$ "*yes*" (where z_3 is computed as z/z_1z_2) and then output z_3 for which the test is satisfied. In this case the security of Gunther's protocol would reduce to the Gap-DH Assumption instead of CDH, but we would not have the Q_2 loss factor.

Test session does not have a matching session For sake of contradiction, let $\mathcal{A}$ be a PPT adversary that is able to break Gunther's protocol with non-negligible advantage ϵ in the case the test session does not have a matching session. Then we show how to exploit such $\mathcal{A}$ to solve the CDH problem with non-negligible probability. S is given in input an instance $(\mathbb{G}, q, g, U = g^u, V = g^v)$ of the CDH problem and wants to output $W = g^{uv}$.

First, S sets the public parameters of the KGC as $MPK = (\mathbb{G}, q, g, y = U, H_1, H_2)$ where H_1 and H_2 are random oracles controlled by S as described below. Let n be an upper bound to the number of all the sessions initiated by $\mathcal{A}$, then the simulator guesses with probability at least $1/n$ the test session chosen by the adversary. We assume Bob to be its peer.

In order to simulate *party corruption* S provides the adversary with access to a key derivation oracle that given in input a user's identity outputs the associated secret key. The random oracle H_1 and the key derivation oracle are programmed as follows. Given in input an identity ID of a user different from Bob ($ID \neq B$), S chooses random $e, d \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $r = g^e y^d$, $s = -rd^{-1}, H_1(ID, r) = es$, outputs (r, s) as private key for the user ID. We assume that the simulator outputs the same answer whenever it is queried on the same input. For the case of Bob the simulator chooses the r_B component of Bob's provate key by picking a random $k_B \overset{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$. It also sets $H_1(B, r_B) = e_B$ for a random $e_B \in \mathbb{Z}_q$. We observe that in this case S is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

For each value $\bar{z}$ received in input, the random oracle H_2 is simulated by choosing a random string $Z \in \{0,1\}^\ell$ and storing the pair $(\bar{z}, Z)$ in a table $\overline{H_2}$.

First we describe how to simulate sessions different from the test session. The simulator has full information about all the users' secret keys except Bob. It turns out that S can easily run protocol sessions and answer to attacker's queries about all parties but Bob. Hence we will concentrate on describing how S simulates interactions with Bob.

Assume that Bob has a session with Charlie. If Charlie is an uncorrupted party this means that S will generate the messages on behalf of him. In this case S knows Charlie's secret key and also has chosen his ephemeral exponents t_C, w_C. Thus it is trivial to see that S has enough information to compute the correct session key.

The case when the adversary presents messages $\langle C, r_C\rangle, \langle u_C, v_C\rangle$ to Bob as coming from Charlie is more complicated. The simulator replies with messages $\langle B, r_B\rangle, \langle u_B = r_C^{t_B}, v_B = g^{w_B}\rangle$ where t_B, w_B are chosen by S. To answer to a session-key query to Bob, S makes use of the Gap-DH oracle as follows. Recall that the session key is $H_2(z_1z_2z_3)$ and that the simulator can compute $z_1 = r_C^{s_C t_B}$ and $z_3 = v_C^{w_B}$ since it knows t_B and w_B. Notice that:

$$r_B^{s_B} = g^{H_1(B,r_B)}y^{-r_B} \text{ and } r_B^{t_C} = u_C.$$

So $z_2 = r_B^{s_B t_C}$ is the Diffie-Hellman result of the two values above. Thus the simulator can check if the table $\overline{H_2}$ contains a tuple $(\bar{z}, Z)$ such that $DH(u_C, g^{H_1(B,r_B)}y^{-r_B}, \bar{z}/z_1z_3) =$ "yes''. If S finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \overset{\$}{\leftarrow} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query $\bar{z}$ to H_2, if $\bar{z}$ satisfies the equation above it answers with ζ. This makes oracle's answers consistent.

Now we describe the simulation of the test session between Alice and Bob, assuming Alice as the owner of the session. If there is no matching session at Bob, it means that the incoming messages $\langle B, \rho_B\rangle, \langle u_B, v_B\rangle$ from Bob to Alice are not sent by an honest party, but they are originated from the adversary. Notice that the adversary may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which S simulated and for which it knows k_B). The simulator replies with the messages $\langle A, r_A\rangle, \langle u_A = V, v_A = g^{w_A}\rangle$ outgoing from Alice, implicitely setting $t_A = v/\lambda_B$. The correct session key for the test session is the hash $Z = H_2(\bar{z})$ where $\bar{z} = u_B^{s_A}(g^{e_B}y^{-\rho_B})^{v/\lambda_B}v_B^{w_A}$. We notice that S is able to compute z_1 and z_3. If the adversary has success into distinguishing Z from a random value it must necessarily query the random oracle H_2 on $\bar{z}$. This means that S can efficiently find $\bar{z}$ in the table $\overline{H_2}$ using the Gap-DH oracle and then exctract $z_2 = \bar{z}/z_1z_3 = V^{e_B/\lambda_B}W^{-\rho_B/\lambda_B}$.

Thus, if A succeeds with advantage ϵ, then S can find z_2 with probability at least ϵ/n. If the adversary has used $\rho_B = r_B$ then S can extract $W = (z_2/(V^{k_B/e_B}))^{-k_B/r_B}$, otherwise it proceeds as follows.

The simulator runs again $\mathcal{A}$ with the same input and the same random coins as in the previous successful run. The only difference is that $H_1(B, \rho_B)$ is answered with a new random value e'_B. Subsequent queries to H_1 are answered with different random values as well.

Changing random oracle responses after $H_1(B, \rho_B)$ can lead $\mathcal{A}$ to make different random tosses after it sees that. Hence we assume that in the test session it creates a message $\langle u'_B, v'_B\rangle$. The adversary may also choose a different peer for the test session. Without loss of generality we continue to call her Alice, but assume she has a different secret key (r'_A, s'_A).

If we apply the result of the Forking Lemma, we obtain that $\mathcal{A}$ is likely to succeed in this "repeat" experiment with non-negligible probability. Thus, using the same procedure as in the first run, S obtains from $\mathcal{A}$ a second value $z'_2 = V^{e'_B/\lambda_B}W^{\rho_B/\lambda_B}$ from which it can compute $\tau = \left(\frac{z_2}{z'_2}\right)^{(e_B - e'_B)^{-1}} = V^{1/\lambda_B}$. In other words we have an algorithm that given in input a pair (g, g^v) is returning in output $(\rho_B = g^{\lambda_B}, \tau)$ such that $\tau = V^{1/\lambda_B}$. If the KEA assumption holds, then there exists an extractor algorithm that given the same input (g, g^u) outputs $(\rho_B, \tau, \lambda_B)$. If S runs such algorithm can obtain λ_B and use it to extract $W = \left(\frac{z_2}{V^{\lambda_B/e_B}}\right)^{-\rho_B^{-1}}$.

Vulnerability to reflection attack In this section we show that Gunther's protocol is vulnerable to the reflection attack. We recall that this attack occurs when an adversary tries to impersonate a party, e.g. Bob to Bob himself. In the case of Gunther's protocol we can restrict this attack to the case when an adversary presents to Bob the first message containing Bob's identity B and the key r_B. In particular, we do not consider the case in which the adversary uses a value $r'_B \neq r_B$ because one can imagine that the honest Bob (who knows his secret key r_B) refuses the connections from himself with $r'_B \neq r_B$.

In this scenario, when (the honest) Bob generates $u_B = g^{t_B}, v_B = g^{w_B}$ and the adversary sends $u'_B = g^{t'_B}, v'_B = g^{w'_B}$ the session key will be $(r_B^{s_B})^{t_B + t'_B} g^{w_B w'_B}$. Thus an adversary, after seeing the message from Bob, can set $u'_B = g^t / u_B$ and $v'_B = g^{w'_B}$ and then is able to compute the session key $H(\bar{z})$ where $\bar{z} = (r_B^{s_B})^t \cdot v_B^{w'_B} = (g^{H(B, r_B)} y^{-r_B})^t \cdot v_B^{w'_B}$.

Other security properties Following arguments similar to those used for protocol IB-KA in Section 4.5, it is possible to show that Gunther's protocol is resistant to KCI attacks and has weak forward secrecy.

D.2 Saeednia's protocol

Saeednia proposed in [22] a variant of Gunther's protocol that allows to reduce to 2 the number of messages exchanged by the parties. The idea of Saeednia was basically to use a different equation for computing the El Gamal signature to generate users' keys. Here we propose a variant of Saeednia's protocol that can be proved secure in the CK model under the Gap-DH assumption. The modified protocol is summarized in Figure 2.

Saeednia's protocol

Setting: A Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q together with a random generator $g \in \mathbb{G}$ and an exponent $x \overset{\$}{\leftarrow} \mathbb{Z}_q$. KGC publishes $\mathbb{G}, q, g, y = g^x$ and two hash functions H_1, H_2. It distributes to each user with identity U a private key (r_U, s_U) computed as follows: $r_U = g^k, s_U = kH_1(U, r_U) + xr_U \bmod q$ for random $k \overset{\$}{\leftarrow} \mathbb{Z}_q$.

Key agreement: A and B choose ephemeral private exponents t_A and t_B, respectively.

$$A \xrightarrow{\quad ID_A, r_A, u_A = g^{t_A} \quad} B$$

$$\xleftarrow{\quad ID_B, r_B, u_B = g^{t_B} \quad}$$

$$z_1 = u_B^{s_A} \qquad\qquad z_1 = (r_A^{H_1(ID_A, r_A)} y^{r_A})^{t_B}$$

$$z_2 = (r_B^{H_1(ID_B, r_B)} y^{r_B})^{t_A} \qquad\qquad z_2 = u_A^{s_B}$$

$$z_3 = u_B^{t_A} \qquad\qquad z_3 = u_A^{t_B}$$

$$Z = H_2(z_1 z_2, z_3)$$

Fig. 2. A and B share session key Z.

We did almost the same modifications proposed for Gunther's protocol in Appendix D.1, namely adding the value r when hashing the identity and hashing the session key. We recall that the session key in the original version of the protocol is the value $z_1z_2z_3$ where z_3 is needed to obtain (weak) FS. In our variant we include z_3 in the hash of the session key as $H_2(z_1z_2, z_3)$.

The following theorem proves the security of the modified Saeednia's protocol.

Theorem 8. *Saeednia's protocol is secure according to Definition 4 under the Gap-DH assumption if H_1 and H_2 are modeled as random oracles.*

Proof. To prove the theorem we distinguish between the case when the adversary chooses a test session that has a matching session and the other case when the test session hash no matching session. In the former we show that we can break the CDH Assumption, while the latter relies on the Gap-DH Assumption.

The test session has a matching session For sake of contradiction assume there exists an adversary $\mathcal{A}$ that is able to break with non-negligible advantage the security of Saeednia's protocol choosing a test session that has a matching session. Then we can build an efficient algorithm S that can solve the CDH problem with non-negligible probability.

S receives in input a tuple $(\mathbb{G}, q, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$. First S simulates the KGC setting up the public parameters of the protocol. It chooses a random $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, q, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$. Similar work is done for H_2.

As it is described in Appendix B the adversary is allowed to ask the KGC for the secret keys of users of its choice (except for the two parties involved in the test session). Hence S must be able to simulate the key derivation process. If the adversary asks for the secret key of a user, the simulator is always able to respond, since it has chosen the master secret key x by itself.

At the beginning of the game S guesses the test session and its holder (let us call him Bob). Also let Alice be the other party of the session. Sessions different from the test session are easily simulated since S knows all the informations needed to compute the session keys and answer to session key queries.

Without loss of generality we assume that the test session is at Bob (and thus the corresponding matching session is at Alice). According to the definition of security, the two parties are uncorrupted at the moment of the test session (i.e. not controlled by the adversary), thus the simulator gets to choose the messages for both them, even for Bob.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity informations and the secret keys of Alice and Bob respectively. The simulator sets Alice's message as $(A, r_A, u_A = U)$ while the one from Bob is $(B, r_B, u_B = V)$. S is implicitly setting $t_A = u, t_B = v$. Since H_2 is modeled as a random oracle, if the adversary has success into distinguishing the real session key from a random value, it must have queried H_2 on the correct input $(z = u_B^{s_A} u_A^{s_B}, z_3 = g^{uv})$. Thus S can choose a random value among all the queries that it received from the adversary. Since the number of queries Q_2 is polynomially bounded, the simulator can find $z_3 = W$ with non-negligible probability ϵ/nQ_2. This completes the proof of this case.

Remark 2. If we would assume the simulator having access to a Gap-DH oracle $\mathcal{O}$, S might use the oracle to test, for all queries (z, z_3) made by the adversary, if $DH(U, V, z_3) =$ "*yes*" and then output z_3 for which the test is true. In this case the security of Saeednia's protocol would reduce to the Gap-DH Assumption instead of CDH, but we would not have the Q_2 loss factor.

Test session does not have a matching session For sake of contradiction, let $\mathcal{A}$ be a PPT adversary that is able to break Saeednia's protocol with non-negligible advantage ϵ in the case when the test session does not have a matching session. Then we show how to exploit such $\mathcal{A}$ to solve the CDH problem with non-negligible probability. S is given in input an instance $(\mathbb{G}, q, g, U = g^u, V = g^v)$ of the CDH problem and wants to output $W = g^{uv}$.

First, S sets the public parameters of the KGC as $MPK = (\mathbb{G}, q, g, y = U, H_1, H_2)$ where H_1 and H_2 are random oracles controlled by S as described below. The simulator also guesses the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

In order to simulate *party corruption* S provides the adversary with access to a key derivation oracle that given in input a user's identity outputs the associated secret key. The random oracle H_1 and the key derivation oracle are programmed as follows. Given in input an identity ID of a user who is not Bob ($ID \neq B$), S chooses random $e, d \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $r = g^e y^d$, $H_1(ID, r) = -rd^{-1}, s = -erd^{-1}$, outputs (r, s) as private key for the user ID. We assume that the simulator outputs the same answer whenever it is queried on the same input. For the case of Bob the simulator chooses the r_B component of Bob's private key by picking a random $k_B \overset{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$. It also sets $H_1(B, r_B) = e_B$ for a random $e_B \in \mathbb{Z}_q$. We observe that in this case S is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

For each pair (z, z_3) received in input, the random oracle H_2 is simulated by choosing a random string $Z \in \{0,1\}^\ell$ and storing each triple (z, z_3, Z) in a table $\overline{H_2}$.

First we describe how to simulate sessions different from the test session. The simulator has full information about all the users' secret keys except Bob. It turns out that S can easily run protocol sessions and answer to attacker's queries about all parties but Bob. Hence we will concentrate on describing how S simulates interactions with Bob.

Assume that Bob has a session with Charlie. If Charlie is an uncorrupted party this means that S will generate the messages on behalf of him. In this case S knows Charlie's secret key and also has chosen his ephemeral exponent t_C. Thus it is trivial to see that S has enough information to compute the correct session key.

The case when the adversary presents a message $\langle C, r_C, u_C \rangle$ to Bob as coming from Charlie is more complicated. The simulator replies with a message $\langle B, r_B, u_B = g^{t_B} \rangle$ where t_B is chosen by S. To answer to a session-key query to Bob, S makes use of the Gap-DH oracle as follows. Recall that the session key is $H_2(z_1 z_2, z_3)$ and that the simulator can compute $z_1 = g^{s_C t_B}$ and $z_3 = u_C^{t_B}$ since it knows t_B. Notice that:

$$g^{s_B} = r_C^{H_1(C, r_C)} y^{r_C} \text{ and } g^{t_C} = u_C.$$

So $z_2 = g^{s_B t_C}$ is the Diffie-Hellman result of the two values above. Thus the simulator can check if the table $\overline{H_2}$ contains a tuple $(\bar{z}, z_3, Z)$ where $DH(u_C, g^{s_B}, \bar{z}/z_1) =$ "*yes*". If S finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \overset{\$}{\leftarrow} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query (z, z_3) to H_2, if (z, z_3) satisfies the equation above it answers with ζ. This makes oracle's answers consistent.

Now we describe the simulation of the test session between Alice and Bob, assuming Alice as the owner of the session. If there is no matching session at Bob, it means that the incoming message $\langle B, \rho_B, u_B \rangle$ from Bob to Alice is not sent by an honest party, but it is originated from the adversary. Notice that the adversary may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which S simulated and for which it knows k_B). The simulator replies with the message $\langle A, r_A, u_A = V \rangle$ outgoing from Alice. The correct session key for the test session is the hash $Z = H_2(z_1 z_2, z_3)$ where $z_1 = u_B^{s_A}$, $z_2 = (\rho_B^{e_B} y^{\rho_B})^v$ and $z_3 = g^{vt_B}$. If the adversary has success into distinguishing Z from a random value it must necessarily query the correct pair $(z_1 z_2, z_3)$ to the random oracle H_2. This means that S can efficiently find the pair $(z_1 z_2, z_3)$ in the table $\overline{H_2}$ using the Gap-DH oracle. Thus, if A succeeds with advantage ϵ, then S can find the pair $(z_1 z_2, z_3)$ with probability at least ϵ/n. If the adversary has used $\rho_B = r_B$ then S can extract $W = (z_2/(V^{k_B e_B}))^{r_B^{-1}}$, otherwise it proceeds as follows.

The simulator runs again $\mathcal{A}$ with the same input and the same random coins as in the previous successful run. The only difference is that $H_1(B, \rho_B)$ is answered with a new random value e'_B. Subsequent queries to H_1 are answered with different random values as well.

Changing the random oracle responses after $H_1(B, \rho_B)$ can lead $\mathcal{A}$ to make different random tosses after he sees that. Hence we assume that in the test session it creates a message $\langle B, \rho_B, u'_B \rangle$. The adversary may also choose a different peer for the test session. Without loss of generality we continue to call her Alice, but assume she has a different secret key (r'_A, s'_A).

If we apply the result of the Forking Lemma, we obtain that $\mathcal{A}$ is likely to succeed in this repeat experiment with non-negligible probability. Thus S obtains from $\mathcal{A}$ a second successful pair (z', z'_3) such that $z' = z'_1 z'_2$ where $z'_1 = g^{t'_B s'_A}$ and $z'_2 = V^{\lambda_B e'_B} W^{\rho_B}$. Then it computes $\tau = \left(\frac{z_2}{z'_2}\right)^{(e_B - e'_B)^{-1}}$ and finally extracts $W = \left(\frac{z_2}{\tau^{e_B}}\right)^{\rho_B^{-1}}$.

Other security properties Saeednia's protocol with the modifications presented above satisfies resistance to KCI and reflection attacks and has weak forward secrecy. To see this, it is possible to observe that the same arguments given in Section 4.5 for the IB-KA protocol apply to this case. In particular, resistance to reflection attacks can be proven under the Square-DH assumption as well, namely we can build an algorithm that computes g^{u^2} when given in input g, g^u.